Programmation Fonctionnelle et Sémantique - Planche de TD 1

Master 1 d'Informatique

2010-2011

Dans chaque exercice on donnera le type des fonctions écrites.

1 Premier ordre

1. Ecrire une fonction *sym* qui à un couple associe son symétrique.
2. Programmer les fonctions booléennes de base : *et*, *ou*, *implique*, *equivaut*, *xor*.
3. Ecrire une fonction qui étant donnés 4 nombres réels a,b, c, et x renvoie la valeur ax^2+bx+c.
4. Ecrire une fonction qui étant donnés 3 nombres réels renvoie la plus grande des deux racines du polynôme $ax^2 + bx + c$ si ce polynôme a deux racines réelles distinctes, et un message d'erreur sinon.
5. Ecrire une fonction qui calcule le PGCD de 2 entiers positifs par l'algorithme d'Euclide.
6. On rappelle que la suite de Fibonacci est définie par :
$$u_0 = 1,\ u_1 = 1,\ \forall n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n$$
Ecrire une fonction qui étant donné un entier n, renvoie le n^{ieme} terme de cette suite.

2 Ordre supérieur

1. (a) Ecrire la fonction *derivee* qui étant donnée une fonction f de $\mathbb{R}$ dans $\mathbb{R}$ et un nombre réel dx, renvoie $\frac{f(x+dx)-f(x)}{dx}$.
   (b) Ecrire la fonction de lissage qui renvoie la valeur moyenne entre f(x), f(x-dx), f(x+dx).
2. Ecrire une fonction *comp* qui prend en argument 2 fonctions et qui renvoie leur composée.

   Par abus de langage on dira qu'une fonction de la forme $(fun\ x\ y\ \rightarrow < expr >)$ *est une fonction à 2 arguments.*

3. Ecrire une fonction sym_func qui, étant donnée une fonction à 2 arguments, renvoie la même fonction mais avec des arguments inversés.
4. Ecrire une fonction *curry* qui étant donnée une fonction à 2 arguments renvoie la curryfiée d'une fonction définie sur un type produit.
5. Ecrire la fonction réciproque *uncurry*.
6. Utiliser des fonctions précédemment définies pour obtenir l'autre curryfiée d'une fonction à 2 arguments.
7. (a) Ecrire une fonction *sigma* qui étant donnés 2 entiers a, b et une fonction f, renvoie : $f(a) + f(a+1) + ... + f(b)$ (si $a <= b$ et 0 sinon) ; ;

(b) Même question pour une fonction *pi* calculant $f(a) * f(a+1) ... * f(b)$ avec f à valeurs réelles. Utiliser cette fonction pour redéfinir la fonction $fact$ qui calcule la factorielle.

(c) Même question pour une fonction pi_2 calculant $f(a) * f(a+2) * f(a+4) * ... * f(a+2n)$ où $a + 2n <= b < a + (2n + 1)$.

(d) Généraliser au cas d'un opérateur *op*, d'une fonction d'incrémentation *incr* , une condition d'arrêt *c* et un élément à renvoyer *e* dans le cas terminal (en général, l'élément neutre de *op*), quelconques. Retrouver *sigma*, *pi* et pi_2.

3 Itérations

– **Itérations bornées**

1. Définir une fonction *iter* qui prend en paramètre une fonction f et un entier n et qui calcule la n^{ieme} itérée de f.
2. Utiliser cette fonction pour calculer la fonction de Fibonacci.

– **Itérations non bornées**

1. Ecrire une fonction *loop* qui itère une fonction f sur un élément x donné tant que x ne vérifie pas un prédicat p. i.e. (*loop p f x*) simule :

```
while not p(x) do x<-f(x);return x
```

2. Retrouver la fonction *iter* à l'aide de *loop*.

4 Dichotomie

Soit f une fonction sur les nombres réels, monotone et continue sur $[a, b]$, telle que $f(a)f(b) < 0$. On sait que la courbe de f coupe l'axe des x i.e. qu'il existe une valeur x_0 telle que $f(x_0) = 0$. On se propose d'approcher la racine en trouvant un intervalle $[a', b']$ qui la contient et tel que $(b' - a') < esp$ ou *eps* est un (petit) réel. Pour cela, on itère le procédé suivant : si m est le milieu de l'intervalle $[a, b]$ on remplace $[a, b]$ par $[a, m]$ si $f(a)f(m) < 0$ et par $[m, b]$ sinon jusqu'a ce que l'intevalle soit de longueur inférieure à epsilon.

Programmer *dicho* qui étant donnés f, a, b et *eps*, renvoie l'intervalle approximant la racine de f dans le cas où elle a les propriétés requises.

Utiliser cette fonction pour trouver une approximation du nombre π.

5 Programmation de l'opérateur de point fixe

1. Rappeler la fonctionnelle $sFact$ dont la fonction factorielle est point fixe.
2. Rappeler les deux définitions possibles de l'opérateur de point fixe fix.
3. Montrer en simulant l'évalutaion Caml de l'expression *(fix sFact)* que l'une de ces définitions est inutilisable.
4. Définir la factorielle comme point fixe de $sFact$ en choisissant la définition correcte de fix et simuler l'évaluation qui se produit au moment de cette définition. En déduire la valeur de la fonction $fact$ ainsi définie.
5. Simuler alors l'évaluation de l'expression *(fact 3)*.